B.5 `pdb` (Python Debugger)

Adapted from https://docs.python.org/3/library/pdb.html.

The module `pdb` defines an interactive source code debugger for Python programs.

The typical usage to break into the debugger from a running program is to insert

```
breakpoint()
```

at the location you want to break into the debugger. You can then step through the code following this statement, and continue running without the debugger using the continue command.

Debugger commands

The commands recognized by the debugger are listed below. Most commands can be abbreviated to one or two letters as indicated; e.g. `h(elp)` means that either `h` or `help` can be used to enter the help command (but not `he` or `hel`, nor `H` or `Help` or `HELP`). Arguments to commands must be separated by whitespace (spaces or tabs).

Entering a blank line repeats the last command entered. Exception: if the last command was a list command, the next 11 lines are listed.

Commands that the debugger doesn't recognize are assumed to be Python statements and are executed in the context of the program being debugged. This is a powerful way to inspect the program being debugged; it is even possible to change a variable or call a function. When an exception occurs in such a statement, the exception name is printed but the debugger's state is not changed.

Command	**Description**
a(rgs)	Print the argument list of the current function.
c(ont(inue))	Continue execution, only stop when a breakpoint is encountered.
h(elp)	Without argument, print the list of available commands. With a *command* as an argument, print help about that command.
l(ist)	List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing. The current line in the current frame is indicated by `->`.

Command	Description
ll	List all source code for the current function or frame. (Short for "long list".)
n(ext)	Continue execution until the next line in the current function is reached or it returns. (The difference between `next` and `step` is that `step` stops inside a called function, while `next` executes called functions at (nearly) full speed, only stopping at the next line in the current function.)
r(eturn)	Continue execution until the current function returns.
s(tep)	Execute the current line, stop at the first possible occasion (either in a function that is called or on the next line in the current function.)